

**OOPS (PRACTICAL) PROJECT ON:**

**APPLIANCE DEVELOPMENT USING JAVA:**

**“electric fan”**

**SUBMITTED TO:**

**“SIR Naveen kumar”**

**BY:**

**ROLL NO:** **24SW032**

**ROLL NO:** **24SW108**

**Electric Fan Simulation:**

**1. Introduction**

This project simulates an **Electric Fan** using Java. It demonstrates **Object-Oriented Programming (OOP)** concepts and **multithreading**. The fan can be turned on/off, its speed can be adjusted, and its status can be shown.   
Two threads run at the same time: One for controlling the fan and Another for displaying the fan's status.

**2. Chosen Appliance and Features**

**Appliance: Electric Fan**

The appliance chosen for this Project is the **Electric Fan**. It’s a common household appliance that circulates air to cool down a room.

**Features of the Electric Fan:**

1. **Turn the Fan On/Off:** The fan can be turned On or Off using the operate() method.
2. **Control the Speed:** You can change the speed of the fan by setting a value using the setSpeed() method.
3. **Display the Status:** The fan's status (on/off, speed, brand) is shown using the showStatus() method.
4. **Multithreading:** The program uses two threads:
   * One for turning the fan on/off and controlling its speed.
   * One for showing the current status of the fan (on/off, speed). 

**3. Implementation of OOP and Threading:**

**OOP Concepts Used:**

1. **Composition:**
   * The **Electric Fan** has a **Motor**. The fan **"has-a"** motor, meaning the fan uses motor to function and MOTOR is strictly attached to the FAN.
2. **Aggregation:**
   * The **Battery** is an example of aggregation because it can exist independently of the fan. The fan needs the battery, but the battery can be removed or replaced.
3. **Inheritance:**
   * The **Electric Fan** class inherits from the **Appliance** class, meaning it is a type of appliance. The Appliance class provides common features (like ON/OFF), and ElectricFan have its specific features like speed control.
4. **Polymorphism:**
   * The operate() method is overridden in the **ElectricFan** class. This means that when the method is called, it behaves differently for an **ElectricFan** compared to any other appliance that might have an operate() method.
5. **Encapsulation:**
   * The fan’s internal data (like whether it’s ON or its SPEED) is kept private. We use **getters** and **setters** to safely access and modify these values from outside the class.

**Threading:**

1. **Thread 1 (Fan Operation):**
   * This thread controls the fan’s motor. It turns the fan ON or OFF and sets the speed.
2. **Thread 2 (Status Display):**
   * This thread shows the current status of the fan (whether it's ON or OFF, and what is its SPEED if its ON).
3. **Synchronization:**
   * The **synchronized** keyword is used to ensure that only one thread can access the fan’s state at a time. This prevents two threads from Changing or Reading the fan’s state at the same time, which could cause errors.
4. **Conclusion:**

In this project, we used **Object-Oriented Programming (OOP)** to model the **Electric Fan** and **multithreading** to simulate the fan's operation and status display. The fan can be turned on/off, its speed can be changed, and its status can be displayed while multiple threads are running. We ensured that these operations happen safely by using synchronization.

